Functional Partitioning: Enabling and Optimizing Exascale Runtime Services

Sudharshan Vazhkudai [*$], Ali Butt [†], John Cobb [*], Xiaosong Ma [*‡], Frank Mueller [‡], Dhabaleswar Panda [§]
[*] Oak Ridge National Laboratory, [†] Virginia Tech, [‡] North Carolina State University, [§] Ohio State University
[$] *Contact Author: vazhkudaiss@ornl.gov*

Motivation: At O(10^5) nodes, O(10^3) cores per node and O(10^9) thread concurrency, creating an exaflop algorithmic architecture that is matched to the given exascale machine architecture design envelope will pose daunting challenges for application scalability, reliability, and end-to-end performance. All of these challenges must be addressed before the exascale machine can deliver on its promise of enabling next-generation science. First, more cores do not automatically translate into more FLOPS experienced by the application. Even on today's platforms, application simulations are unable to efficiently scale to the available intra-node core counts, so much so that even leaving a core/node idle does not impact performance. In fact, modern node architectures such as the IBM BG/Q (with 18 cores/node) are designed to reserve a core for the OS and another is left as spare. Second, designing simulations for massive-cores per node requires application developers to monitor, tune and debug at scale to derive optimal performance. In addition, a reliable application run will require users to plan and recover from failures that will be the norm in future machines. Finally, delivering exaflop performance is one thing; delivering exascale science is an altogether different challenge. We can no longer afford to look at the application performance in terms of FLOPS alone. Rather, the science relevant metrics are a combination of FLOPS, reliability and scalability within an acceptable energy envelope. Exascale facilities will not reach their full potential if these problems are not resolved. New paradigms are needed to scale application performance in future massive-core systems for the exascale era and beyond.

Solution: Our proposal is centered on the observation that in the post-petascale environment, everything off the compute node is too far away in terms of "clocks." Many system resources and tools will need to be present (or at least represented) on a compute node and integrated and composed into system-level services at scale. Providing required resources outside of the node will not scale when we have O(10^5) nodes, each with 10^3 cores. Just as computational tasks are hierarchically subdivided and distributed across nodes and cores, other services required for effective simulation performance need to be hierarchically subdivided in a scalable and distributed fashion across the computational elements in a manner that facilitates overall application performance. Such distributed, scalable services include data placement, micro-task orchestration, monitoring, resiliency, validation, deep-memory access, etc. In the multi-petaflop and exaflop era, applications are likely to be so complex that we need on-node, lightweight, resilient, in-job resource allocation and management to better provision the available cores to various application activities. We propose a new computing paradigm for massive-core machines. Rather than dedicating all the cores on the compute nodes to only execute the homogeneous, parallel application simulation, we propose a **functional partitioning (FP) runtime service environment** for scalable performance by dedicating a subset of cores within a compute node to perform heterogeneous, application support activities, as a generalized approach to scalable and reliable execution at extreme-scale. With FP, we envision an environment, where each compute node hosts internal services (servlets) for different system functions, making the node itself a fundamentally scalable unit for service composition. The goal here is to define ways to view and organize the servlets (functions performed on a node) so that they can be composed into larger assemblies, while maintaining scalable performance, up to millions of cores and beyond, to be used in addressing many parts of the computing challenge. While the majority of cores per node (compute cores) are still occupied by the main applications simulation execution (MPI+OpenMP [1-4] or UPC [5], etc.), a subset of cores is dedicated to a variety of application support services. These *servlets work in concert with the main computation*, playing supporting or consumer roles. Thus, thematic to FP are the core ideas of *decentralizing* critical exascale services by placing them as "servlets" within each node and then *spatially multiplexing* them within the node to functionally distinct cores, alongside the application's main computation. We argue that such an approach achieves scalability and performance at the very fabric of exascale architectures, making the socket/node into a *self-contained* entity. The analogy being: just as a large machine has compute, I/O and service nodes for different functions, we will enable a "system *on chip"-like* design by partitioning the cores in a node based on their functionality: compute, monitoring, resilience, data placement, networking and communication, efficient access to deep memory and analytics cores, etc. This is an inevitable—but often under-appreciated—requirement for many-core systems that can co-exist with MPI, OpenMP, etc. *We argue that such an outlook brings a fresh perspective to current multi-core runtime research, the vast majority of which is focused only on raw scaling of applications.*

Interference Control: In our proposed environment, although the simulation and the FP servlets are run on separate cores many servlets will end up sharing data with the simulation. We propose to investigate such contention in three areas: main memory consumption, cache, and interconnection bandwidth sharing. We will create FP services to reduce interference. The impact of FP services, themselves, on interference is small and can be minimized using techniques such as out-of-core execution, i.e., on deep-memory NVM tiers (e.g., using our ongoing work on NVMalloc [6] for out-of-core data analytics.) While these accesses will be slower than DRAM latencies, it is a viable alternative to reduce the memory footprint while also making considerable progress on the FP servlets and, therefore, on the application's overall effective performance. Next, we propose to coordinate the simulation and FP services in order to minimize contention to the interconnection bandwidth. An application can explicitly tell the FP services to hold-off any network traffic and buffer data until the main simulation is done with its transfers and communication traffic. We will explore this ability to define *back off periods*, which is the time when FP services should simply buffer their data and not utilize the network. For shared last-level cache contention, we argue that FP services are not cache-intensive and in many cases are synergistic with the main simulation. However, even when they are not, we can use prior research and tools such as ULCC [7] and other cache partitioning work [8] that can help minimize cache contention. One research issue pertinent to FP that needs to be addressed is how to determine and tune the cache partition sizes so that FP services do not lag behind in processing the data produced by the simulation.

Related Work: There have been studies on utilizing available cores for online execution monitoring [9-11]. Several research efforts have also advocated a pipelined model — that assigns various computational tasks of an application to different cores for [12-16] for parallelizing applications. There exist I/O libraries that dedicate processors or threads [17-19] for handling parallel I/O operations. Our FP approach for multicores directly targets the on-chip parallel computation efficiency problem, and presents a more general and versatile service model for balanced utilization of the increasing number of cores. Instead of having multiple core-specialization services developed independently, FP proposes a framework within which these services: can be built, interact with the main application, can share on-chip resources in a coordinated fashion, co-exist with other core-specialization services and work in concert with the application towards effective performance. To the best of our knowledge, a generic runtime framework to functionally partition the available cores has not been studied for mainstream HPC applications. Efforts such as the Tessellation OS partition manager [20] and factored OS (fos) [21] attempt to build scalable operating systems for massive-cores. It guarantees partitions of on-node physical resources to disparate, applications (often competing) and OS services. Factored OS proposes a distributed set of servers for each OS kernel service and assigns them to different cores for scalability. The goal there is to build OS services that run on different partitions from the applications. Instead, the FP runtime is concerned with the spatial scheduling of an exascale application's own services and can work atop these operating systems. The services are not competing applications, but are intended to work in concert with the application's simulation component, achieving better overall end-to-end performance. The FP runtime is responsible for this coordination and controlling any interference.

Challenges addressed: FP will improve application turnaround, scalability, run reliability, energy consumption by minimizing data movement, hiding latencies, and alleviating the pressure on storage subsystems by reducing output sizes.

Maturity and indicators that the approach will address the challenges: FP attempts to achieve a fundamental scalability at the socket/node-level, enabling larger assemblies. Our preliminary work on multiplexing data services alongside simulation in Supercomputing 2010 suggests that FP can not only improve application turnaround, but also reduce data movement costs [22].

Uniqueness: In the exaflop era, applications are likely to be so complex that we need on-node, lightweight, resilient, in-job resource allocation and management to better provision the available cores to various application activities. Thus, FP is unique to exascale and is not likely to be addressed by other programs.

Novelty: Allocating multiple services on a set of cores is different from most other current exascale efforts that restrict themselves to only considering collections of cores, each of which are dedicated to one service. Striping of closely coupled services across cores enables higher levels of scalability and reduces remote data placement demands. Finally, FP posits a novel design philosophy to runtime construction, one that is based on service composition rather than monolithic techniques.

Effort: We foresee that a four-year, multi-institutional effort will be required to effectively explore the FP paradigm. A bottom-up exercise of identifying key exascale services, decentralizing them into on-node servlets, and co-locating them with simulations is needed to understand the interplay between them. Such an approach is critical to identifying and designing the functionality of the FP runtime framework.

References

[1] Message Passing Interface Forum, *MPI: Message-Passing Interface Standard,* 1995.

[2] OpenMP, OpenMP Application Program Interface, v. 3.0, http://www.openmp.org, 2008.

[3] L. Dagum and R. Menon, OpenMP: An Industry-Standard API for Shared-Memory Programming, *IEEE Computational Science and Engineering* 5 (1998).

[4] A. Scherer, H. Lu, T. Gross, and W. Zwaenepoel, Transparent Adaptive Parallelism on NOWs using OpenMP, in *Principles Practice of Parallel Programming,* 1999.

[5] Berkeley UPC - Unified Parallel C, http://upc.lbl.gov/, 2011.

[6] C. Wang, S.S. Vazhkudai, X. Ma, F. Meng, Y. Kim, C. Engelmann, "NVMalloc: Exposing an Aggregate SSD Store as a Memory Partition in Extreme-Scale Machines", *Proceedings of the 26th IEEE Int'l Parallel & Distributed Processing Symposium (IPDPS 2012)*, Shanghai, China, May 2012.

[7] Xiaoning Ding, Kaibo Wang, and Xiaodong Zhang, ULCC: A User-Level Facility for Optimizing Shared Cache Performance on Multicores, *Proceedings of 16th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming* (PPoPP 2011), San Antonio, Texas, February 12-16, 2011.

[8] J. Lin, Q. Lu, X. Ding, Z. Zhang, X. Zhang, and P. Sadayappan. Gaining Insights into Multicore Cache Partitioning: Bridging the Gap between Simulation and Real Systems. *In Proceedings of International Symposium on High Performance Computer Architecture (HPCA 2008)*, pages 367--378, 2008.

[9] Devesh Tiwari, Sanghoon Lee, James Tuck, and Yan Solihin, MMT: Exploiting Fine-Grained Parallelism in Dynamic Memory Management, *Proceedings of IPDPS*, Atlanta, GA, 2010.

[10] Shimin Chen, Babak Falsafi, Phillip B. Gibbons, Michael Kozuch, Todd C. Mowry, Radu Teodorescu, Anastassia Ailamaki, Limor Fix, Gregory R. Ganger, Bin Lin, and Steven W. Schlosser, Log-based architectures for general-purpose monitoring of deployed code, in *Proceedings of the 1st Workshop on Architectural and System Support for Improving Software Dependability,* pages 63–65, 2006.

[11] Edmund B. Nightingale, Daniel Peek, Peter M. Chen, and Jason Flinn, Parallelizing security checks on commodity hardware, in *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems,* pages 308–318, 2008.

[12] Kue-Hwan Sihn, Baik Hyunki, Kim Jong-Tae, Bae Sehyun, and Song Hyo Jung, Novel approaches to parallel H.264 decoder on symmetric multicore systems, in *Proc. IEEE ICASSP,* 2009.

[13] S. Arash Ostadzadeh, Roel J. Meeuws, Kamana Sigdel, and Koen Bertels, A Multipurpose Clustering Algorithm for Task Partitioning in Multicore Reconfigurable Systems, in *Proc. IEEE CISIS,* 2009.

[14] Turgay Altilar and Yakup Paker, Minimum Overhead Data Partitioning Algorithms for Parallel Video Processing, in *Proc. International Conference on Domain Decomposition Methods,* 2001.

[15] Robert Ennals, Sharp, and Mycroft, Task Partitioning for Multi-core Network Processors, in *Proc. Cluster Computing,* 2005.

[16] Sanjay Kumar, Gavrilovska, Karsten Schwan, and Srikanth Sundaragopalan, C-CORE: Using Communication Cores for High Performance Network Services, in *Proc. IEEE NCA,* 2005.

[17] Xiaosong Ma, Jonghuyn Lee, and Marianne Winslett, High-level Buffering for Hiding Periodic Output Cost in Scientific Simulations, *IEEE Transactions on Parallel and Distributed Systems* 17 (2006).

[18] Kent E. Seamons, Ying Chen, P. Jones, J. Jozwiak, and Marianne Winslett, Server-directed collective I/O in Panda, in *Proceedings of Supercomputing '95,* 1995.

[19] Sachin More, Alok Choudhary, Ian Foster, and Ming Q. Xu, MTIO: A Multi-Threaded Parallel I/O System, in *Proceedings of the Eleventh International Parallel Processing Symposium,* 1997.

[20] Rose Liu, Kevin Klues, Sarah Bird, Steven Hofmeyr, Krste Asanovi, and John Kubiatowicz, Tessellation: Space-Time Partitioning in a Manycore Client OS, in *Proc. of HotPar09,* 2009.

[21] David Wentzlaff and Anant Agarwal, Factored Operating Systems (fos): The Case for a Scalable Operating System for Multicores, *ACM SIGOPS Operating System Review (OSR)* (2009).

[22] Min Li, Sudharshan S. Vazhkudai, Ali R. Butt, Fei Meng, Xiaosong Ma, Youngjae Kim, Christian Engelmann, and Galen M. Shipman, Functional Partitioning to Optimize End-to-End Performance on Many-core Architectures, *Proceedings of Supercomputing 2010 (SC10): 23rd IEEE/ACM Int'l Conference on High Performance Computing, Networking, Storage and Analysis*, New Orleans, Louisiana, November 2010.